Matrícula (First-name Last-name) **CK0030: Homework 04**

Exercise 1. We want to determine the volume V occupied by some gas at some temperature T when subjected to some pressure p. The state-equation, the equation that relates p, V and T, is

$$\left[p + w_1(N/V)^2\right](V - Nw_2) = NkT, \tag{1}$$

in which w_1 and w_2 are coefficients that depend on the specific gas, N is the number of molecules contained in the volume V and $k = 1.3806503 \cdot 10^{-23}$[Joule K^{-1}] is the Boltzmann constant.

The value of V can be obtained by finding the roots of the nonlinear equation $f(V) = 0$, in which

$$f(V) = pV + w_1\frac{N^2}{V} - w_1w_2\frac{N^3}{V^2} - pNw_2 - NkT.$$

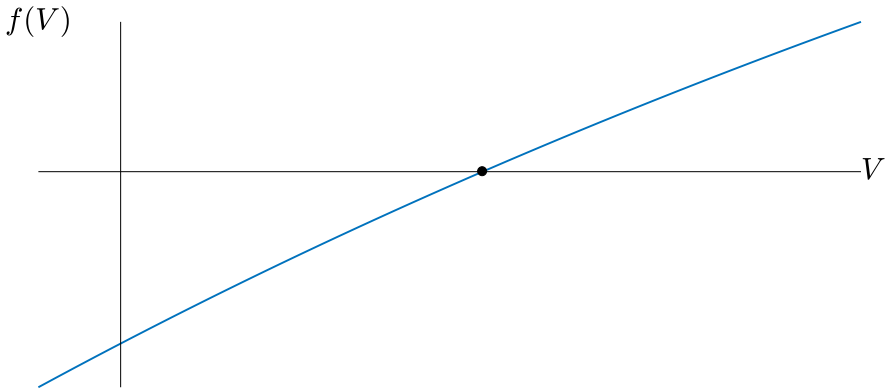

Figure 1: A sketch of function $f(V)$ over some small interval around its zero.

As we cannot compute the zeros of a generic (non-polynomial) function as $f(V)$ in a finite number of operations, an iterative solution must be devised. One such strategy is the bisection method. The bisection method will be our numerical tool for obtaining an approximate solution $\hat{V}$ to the problem.

- You are requested to write a Python program (say, `my_molecules.py`) for the task.

The experimental conditions are p ($3.5 \cdot 10^7$ [Pa]), T (300 [K]), and N (1000 [number of molecules]). The values of the coefficients w_1 (0.401 [Pa m^6]) and w_2 ($42.7 \cdot 10^{-6}$ [m^3]) for Carbon Dioxide (CO_2) are available for download from this `file`. [*Note*]: Pa denotes Pascal and K denotes degrees Kelvin.

The call to execute the program must be of the general form

```
Terminal > my_molecules --f "Function_expression" --d "Datafile_name"
                        --T T_value --p p_value --N N_value
                        --a a_value --b b_value --e e_value
```

This means that $f(V)$, `Datafile_name`, T, p N and the parameters of the bisection method (a, b and ε) must be provided as command-line arguments by the user as option-value pairs.

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--name', '--long_name', type=obj_type,       # Sample argument
                    default=default_value, ... )                  # definition
args = parser.parse.args_args()
```

Inside `my_molecules.py`, function $f(V)$ must be implemented as a Python function defined from `Function_expression`. You must use function `exec` to turn the input string into a Python function.

```
code = """
def f(x):
 return %s
""" % expression
exec(code)
```

The bisection method, as described in the next page, must be implemented in your program as a Python function (say, `my_bisection(f,a,b,e)`) that takes as its input arguments $f(V)$, the lower and upper search limits a and b, and a tolerance ε value and returns the zero ($V = \hat{V} = \alpha$) of $f(V)$.

```
def my_bisection(f, a, b, e)
 ...
 return alpha
```

Instructions

[*Deadline*]: Submissions via SIGAA close Sunday July 02, 2017 at 23:59:59 (Fortaleza Time).

[*Delays*]: Delayed submissions via email to `rafa.olv.lima@gmail.com`. Delays will be penalised.

[*Solutions*]: You can write your solutions in either Portuguese or English language. Solutions must be submitted as PDF or Plain Text files (that includes `.ipynb` and `.py` file formats); Other formats (`.doc`, `.docx`, `.rtf` etc.) will not be considered. The LaTeX template available at the course website is recommended, though not obligatory.

[*About code*]: If the code is short (i.e., at most 3-page long, overall), it is okay to paste it to your solution sheet. Otherwise, it is more appropriate to either package it (`zip`) together with your solution sheet, or provide a link in your submission for us to download it (Note: If you opt for the link, it is your responsibility/risk to make sure that the link is fully functioning also after deadline.)

[*Others*]: Collaborations and solutions inspired by other people's work will be tolerated only within the limits explained in the website. Plagiarism will not be tolerated and will be reported to the UFC.

The bisection method

The bisection method exploits the fact that any function $f(x)$ continuous in $[a, b]$ admits at least one zero in (a, b), if $f(a)f(b) < 0$. Given an initial interval, the method halves it and, among the two resulting sub-intervals, it selects the one within which function f changes its sign. The procedure is then repeated/iterated on the selected sub-interval, until a stopping criterion is met.

From the figure, let $I^{(0)} = (a, b)$ be the initial interval and, in general, let $I^{(k)}$ be the sub-interval selected at the k-th iteration. Interval $I^{(k+1)}$ is chosen as one of the two semi-intervals of $I^{(k)}$ in which f changes its sign. The procedures guarantees that the zero α is found in each interval $I^{(k)}$.

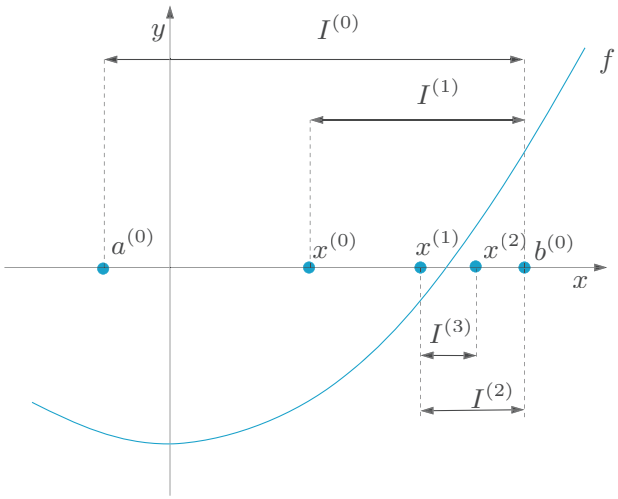

Figure 2: A few iterations of the bisection method on some arbitrary function $y = f(x)$.

More formally, let $a^{(0)} = a$, $b^{(0)} = b$, $I^{(0)} = (a^{(0)}, b^{(0)})$, and $x^{(0)} = (a^{(0)} + b^{(0)})/2$. For $k \geq 1$, calculate the interval $I^{(k)} = (a^{(k)}, b^{(k)})$ from interval $I^{(k-1)} = (a^{(k-1)}, b^{(k-1)})$ as follows.

Given $x^{(k-1)} = (a^{(k-1)} + b^{(k-1)})/2$ and some small number ε

If $f(x^{(k-1)}) \leq \varepsilon$, then $\alpha = x^{(k-1)}$ and the iterations stop

Otherwise (i.e., $f(x^{(k-1)}) > \varepsilon$)

If $f(a^{(k-1)})f(x^{(k-1)}) < 0$, let $a^{(k)} = a^{(k-1)}$ and $b^{(k)} = x^{(k-1)}$
If $f(x^{(k-1)})f(b^{(k-1)}) < 0$, let $b^{(k)} = b^{(k-1)}$ and $a^{(k)} = x^{(k-1)}$

Then, $x^{(k)} = (a^{(k)} + b^{(k)})/2$ and let k increase by 1.